RINGO – Block Based Algorithm for Large Terrain Rendering

DEJAN RANČIĆ, ALEKSANDAR DIMITRIJEVIĆ, BRATISLAV PREDIĆ
Department of Computer Science, Faculty of Electronic Engineering
University of Niš
Aleksandra Medvedeva 14, 18000 Niš
SERBIA

Abstract: – Nowadays, the 3D terrain rendering algorithms have very important role in the wide variety of applications, starting from the video games up to the time-critical simulations, and real-time systems for command and control. RINGO algorithm presented in this paper enables the out-of-core terrain rendering with minimal CPU utilization. Based on OpenGL, it is easily portable to various platforms and operating systems.

Key-Words: – terrain rendering algorithm, out-of-core, level-of-detail, OpenGL, block based organization

1 Introduction
The terrain rendering algorithms is not a new topic in the computers world. Some early works in terrain simplification dated from late 1970s [1], but the real explosion of algorithms started in 1990s [2-4]. The main problem that all of the algorithms tried to solve was to optimize the number of graphics primitives representing a terrain. Generally, there are two approaches in terrain simplification:
• Top-down – starting with two or four triangles which have to be progressively tessellate, until desired resolution is achieved [3,4], and
• Bottom-up – starting with highest-resolution mesh that has to be iteratively simplified, until desired resolution is achieved [2].

The data structures used to represent the terrain can be: regular gridded height fields or triangulated irregular networks (TINs). TINs achieve required accuracy with fewer polygons, but they are much harder for manipulation, comparing to regular grids. Furthermore, today graphics processing units (GPUs) easily handle large data sets with regular (uniform) structures, so many algorithms rather relay on regular gridded data. The optimal feeding of graphics pipeline is now more important than the fine tessellation tuning.

Our work was very much influenced by the algorithm called Geometry Clipmaps [5], which caches the terrain in a set of nested regular grids centered about the viewer. An important aspect of this approach is that the level of detail (LOD) is independent of data content, and therefore the terrain data does not require any precomputation of refinement criteria.

All previously mention algorithms deal with terrain data that is already loaded into main memory. But, it is fairly desirable feature to enable out-of-core operation. This means that the algorithm should be able to browse terrain data set that exceeds the size of the available main memory. There are many proposed solution for optimizing terrain data layout on the hard disk drive, and improvement of spatial coherence [6-8].

In this paper we present our RINGO algorithm for the out-of-core large terrain rendering. The solution is developed in the CG&GIS Laboratory, at Faculty of Electronic Engineering in Niš, and intended for the implementation in the heavily loaded Geographic Information Systems (GIS). The main requirements for the algorithm are:
• low CPU utilization,
• highest possible rendering rate, and
• portability.

The paper is organized as follows. The second section describes why the display lists are used as base rendering technique for the algorithm, the third section gives the RINGO algorithm overview, and the fourth section discuss various LOD schemes and their influence on data coherency and the speed of moving through the terrain.

2 Examining OpenGL Rendering Techniques
The greatest problem in all terrain simplification algorithms is their CPU utilization. Our goal was to find out the technique that will enable large terrain rendering with minimal CPU load and
implementation in system already heavily loaded with information processing.

For the purpose of implementing the rendering engine, we choose OpenGL as cross-platform powerful graphics library. In order to avoid compatibility problems, only basic functionality is used.

To choose the best method for rendering, a test application has been developed. The test application simulates flying through the terrain that consists of 2.88 million triangles and uses different techniques for data organization (triangles, triangle strips, vectors, display lists and vertex arrays). The speed test is carried out on various computer configurations. Some of them were equipped with powerful graphics accelerators, while the others were with integrated ones. Table 1 presents all configurations used in the test.

<table>
<thead>
<tr>
<th>Label</th>
<th>Graphics</th>
<th>CPU</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>MX440</td>
<td><a href="mailto:P4@2.6GHz">P4@2.6GHz</a></td>
<td>XP</td>
</tr>
<tr>
<td>R2</td>
<td>Radeon 9200</td>
<td><a href="mailto:P4@2.6GHz">P4@2.6GHz</a></td>
<td>XP</td>
</tr>
<tr>
<td>R3</td>
<td>Intel 810 int.</td>
<td>Celeron2@1GHz</td>
<td>XP</td>
</tr>
<tr>
<td>R4</td>
<td>Radeon 9250</td>
<td><a href="mailto:P4@2.6GHz">P4@2.6GHz</a></td>
<td>XP</td>
</tr>
<tr>
<td>R5</td>
<td>Intel Montara</td>
<td><a href="mailto:Centrino@1.5GHz">Centrino@1.5GHz</a></td>
<td>W2K</td>
</tr>
<tr>
<td>R6</td>
<td>Fx5500</td>
<td>Celeron2@800MHz</td>
<td>W98</td>
</tr>
<tr>
<td>R7</td>
<td>Radeon 9700</td>
<td><a href="mailto:Centrino@1.6GHz">Centrino@1.6GHz</a></td>
<td>XP</td>
</tr>
<tr>
<td>R8</td>
<td>Nv6200</td>
<td><a href="mailto:P4@2.8GHz">P4@2.8GHz</a></td>
<td>XP</td>
</tr>
<tr>
<td>R9</td>
<td>Intel 915G int.</td>
<td><a href="mailto:P4@3.0GHz">P4@3.0GHz</a></td>
<td>W2kS</td>
</tr>
<tr>
<td>R10</td>
<td>Nv6600</td>
<td>Athlon64 3000+</td>
<td>XP</td>
</tr>
<tr>
<td>R11</td>
<td>Nv7800</td>
<td>Athlon64 3000+</td>
<td>XP</td>
</tr>
<tr>
<td>R12</td>
<td>Radeon x850 Pro</td>
<td><a href="mailto:P4@3.0GHz">P4@3.0GHz</a></td>
<td>XP</td>
</tr>
</tbody>
</table>

Table 1. Configurations used in the speed test

The averaged and normalized results of the tests are shown in Fig.1. Rendering with TRIANGLES primitives, as it was expected, gives the worst results. TRIANGLE_STRIPS accelerate rendering by reducing function-calls for defining vertex spatial coordinates, texture coordinates, normals and other parameters of the mesh. Theoretically, speed should be boosted up to 3 times, but in the practice it is slightly below that. Further optimization can be obtained by organizing data in vector, replacing multiplication with addition and even reducing number of additions. Although the number of arithmetical operations is significantly reduced, performance gain is pretty poor. This means that arithmetical operations have little influence on overall score, especially in the systems with slow graphics cards.

According to specifications, vertex arrays should also boost rendering speed, but the results of the tests are very controversial. On several configurations results are even worse than triangle strips. So, the display lists are the best solution for boosting the rendering speed. On all configuration display lists achieved better results than any other technique. If the system has any bottleneck, for example slow main memory, system bus or AGP port, using display lists can be up to 20 times faster than the TRIANGLES primitives. That was the case on the configuration R6, where old motherboard and AGPx2 limit the speed of Fx5500. It is obvious that only the display lists can release the full power of the graphics card, but if and only if the whole scene can reside in its memory. Because of such results, to fulfill requirement for highest rendering rate, we chose display lists as the basic technique for the large terrain rendering.

3 Algorithm Overview
The display lists usage has same drawbacks. For example:

- Building the display list is time consuming (and, as we will see, it is not negligible), and
- Display lists reside in the graphics card memory and cannot be changed (just destroyed and rebuilt).

![Fig.2 Terrain segmentation into blocks of equal spatial size](image-url)
Although these drawbacks disable some advanced techniques, such as vertex morphing, minimization of CPU load and maximization of speed justifies the use of display lists.

In order to achieve better utilization of OpenGL display lists, as well as partially loading terrain data, the whole terrain that resides in main memory is divided into blocks (Fig.2). Each block has the same spatial size, but resolution depends upon the distance from the viewer. Also, each block is represented with just one display list.

All terrain blocks are organized into one central area and the certain number of rings (Fig.3). The central area covers 3×3 blocks of the highest resolution, and each ring consists of blocks with four times less triangles than their inner neighbors. To prevent T-junction cracks, borders of some blocks contain additional triangle lists.

Fig.3 The ring organization of terrain blocks

One of the greatest advantages of this algorithm is that it enables high level of parallelism. Namely, loading and preparing each block is totally independent of others. The other advantage is simplicity. The viewer should always be in the central block, and while he is inside the borders of the central block nothing is changing. Moving through the central block enables highest possible frame rate and almost zero CPU utilization.

Fig.4 Viewer crossing the border of the central block

When viewer crosses the border of the central block, blocks reorganization starts. Some of the blocks have to be re-created (in Fig.4 blocks marked with ‘R’), while others just change their position in the terrain matrix (by pointers moving). Blocks recreation is the only time-critical operation in our algorithm. The number and the resolution of re-created blocks directly determine the fluidness of the transition. Although, the time for the blocks creation depends whether the blocks resides in front or behind the viewer (blocks that are behind the viewer can reuse data from their neighbors with the higher resolution), for the purpose of simplification of the following discussion, we will assume that the creation time for any block is the same.

The process of block creation has the following steps:

- Load BIL,
- Build XYZ Matrix,
- Build Color Matrix or Initialize Texture,
- Build Normal Matrix (optionally), and
- Initialize Display List.

The time consumption of each step depends on many factors, such as: CPU and memory speed, size of the blocks, texture compression algorithm, etc. Fig.5 shows typical contribution of each step in the total block creation time.

Fig.5 Typical contribution of each step in the block loading time

The purpose of the LoadBIL step is to find and retrieve required terrain heights from DEM (Digital Elevation Model) or BIL (Band Interleaved by Line) files. Keeping the file chunks small, i.e. less than 200 KB per chunk enables efficient caching and very fast data retrieval.

As it is shown on the Fig.5, Building of XYZ Matrix is the most time consuming operation. This is because of the fact that the algorithm performs a very precise calculation for each vertex. The calculation consists of:

- Earth radius calculation according to WGS84 ellipsoid,
- Adding correction for the geoid,
- Adding height from digital elevation model (loaded at previous step), and
Polar to Cartesian coordinates transformation.

The time for this step can be reduced by performing full calculation only for the small number of vertices, and then interpolating values for the rest. If blocks serve not just for the terrain visualization purpose, but also for some other calculation, interpolation must be carried out very carefully, or has to be completely avoided.

The next step in our algorithm is building color matrix or initializing texture for the block. Building color matrix is also time-consuming operation, because it requires color calculation according to terrain height and slope at each vertex.

Initializing display list is the final and the only step that cannot be optimized. So, theoretically, with all optimization, this algorithm cannot be even 10 times faster than it is now. Due to the parallelism that is inherent feature of our algorithm, each block is created in the separate thread. The blocks that are farther from the viewer, because of their lower resolution, are created faster, enabling the horizon consistency.

4 Coherency and Speed

Every out-of-core algorithm requires slow data access or data creation. To obtain better performance, algorithm should take advantage of:

- Locality - just a small, but sufficient part of data collection should be presented in the main and/or graphics memory, and
- Reusability - maximize the use of data already in the main and/or graphics memory.

In this section we will explain how the width of the rings affects reusability of the blocks and the blocks creation time.

Depending on chosen LOD scheme, blocks can be replaced or moved much or less easy. In the scheme where the rings are of the same spatial width, and the resolution of the blocks in the outer rings is four times less then the resolution of the neighboring blocks in the inner ring (LOD scheme 1), the viewer moving across the border of the central block can require about 6\(T_0\), where \(T_0\) is the time needed for re-creation the block of highest resolution. In the special case, crossing central block through its corner, number of blocks that have to be re-created is about two times more. In all following equations and comparing results, we will assume that the viewer is crossing the edge of the block, which is more usual case than the corner crossing.

The block has to be re-created if and only if it does not stay in the same ring while moving in the opposite direction of the viewer movement. According to Fig.4 the time needed for all those blocks re-creation can be expressed by equation (1). If we assume that re-creation time directly depends on the resolution of the block, and that the resolution of the outer blocks is four times less than resolution of their inner neighbors (2), then the total time for blocks re-creation can be expressed by (3). For the infinite number of rings, the total time is 6.111 times greater than the re-creation time of one block with highest resolution (4).

\[
\begin{align*}
1) \quad T_S^1 &= 3T_0 + (5+3)T_1 + (7+5)T_2 + (9+7)T_3 + ... \\
2) \quad T_{i+1} &= \frac{1}{4}T_i \\
3) \quad T_S^1 &= 3T_0 + \sum_{i=1}^{n} \frac{i+1}{4^{i-1}} T_0 \\
4) \quad T_S^1 &\approx 6.111T_0
\end{align*}
\]

Table 2 summarizes number of blocks (All) in central area (CA) and first three rings (1, 2 and 3), number of re-created and reused block and their participation in total number of blocks of certain resolution.

<table>
<thead>
<tr>
<th>Ring</th>
<th>Recreated</th>
<th>Reused</th>
<th>All</th>
<th>% Recreated</th>
<th>% Reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>33.3</td>
<td>66.7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>12</td>
<td>24</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>50.0</td>
<td>50.0</td>
</tr>
</tbody>
</table>

Table 2 Reusability of the blocks in LOD scheme 1

If each ring is two blocks wide (LOD scheme 2), the reusability of the blocks will increase (Table 3). The total time needed for blocks re-creation also increases (7).

\[
\begin{align*}
5) \quad T_S^2 &= 3T_0 + (7+3)T_1 + (11+7)T_2 + (15+11)T_3 + ... \\
6) \quad T_S^2 &= 3T_0 + \sum_{i=1}^{n} \frac{8i+2}{4^{i-1}} T_0 \\
7) \quad T_S^2 &\approx 7.111T_0
\end{align*}
\]

Table 3 Reusability of the blocks in LOD scheme 2
Further rings widening will increase reusability. For the very large terrains, with far away horizon, it is very useful to double the width of each ring as moving farther from the viewer (LOD scheme $2w$).

**Fig.5 Position of the re-created blocks (marked with 'R') in the terrain matrix with LOD scheme 2**

**Fig.6 Block organization in the terrain matrix with LOD scheme $2w$**

LOD scheme $2w$ requires only 6.67 $T_0$ for all blocks re-creation, which is slightly worst result than LOD scheme 1, but the terrain matrix covers significantly larger spatial area. Reusability factors of this scheme are given in Table 4.

<table>
<thead>
<tr>
<th>Ring</th>
<th>Recreated</th>
<th>Reused</th>
<th>All</th>
<th>% Recreated</th>
<th>% Reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>33.3</td>
<td>66.7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>42</td>
<td>56</td>
<td>25.0</td>
<td>75.0</td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>182</td>
<td>208</td>
<td>12.5</td>
<td>87.5</td>
</tr>
</tbody>
</table>

**Table 4 Reusability of the blocks in LOD scheme 2w**

Generally, LOD schemes with higher reusability factor are better, because they enable larger terrain representation, but the only factors that determine system performances are total re-creation time and total numbers of blocks in the terrain matrix.

5 Conclusion

The block based terrain-rendering algorithm – RINGO, presented in the paper, offers three main advantages: maximal frame-rate that the graphics card can achieve, minimal CPU usage when the viewer remains in the central block, and high parallelism in blocks re-creation procedure. The maximum frame rate and minimum CPU utilization can be achieved if and only if all terrain blocks can reside in the memory of the graphics card. In other case, for example when algorithm is executed on integrated video cards, performances are very poor. The spatial block size should be chosen according to application of the algorithm. If the viewer has a small radius of movement it is wise to choose large blocks to prevent central block border crossing event, and avoid blocks re-creation. But, if the application requires the fast viewer moving, then smaller blocks give better performance. It is also important to keep high the blocks reusability factor when choosing LOD scheme, because it has a direct influence on the fluidness of the viewer crossing the blocks borders. This algorithm is still under development, and certainly requires coarser or finer tuning in many aspects, but currently results promise successful usage in various applications.

References:


